**Use Cases - Strings**

The Redis string data structure is the most versatile data structures that can be used across multiple use cases;

* **For serving static websites pages.** Redis.io site uses strings to serve all static page contents
* **Caching** - To store most common, frequently used data within an application or a website.
* **Counters** are excellent choice for strings e.g Daily website visitors and more.
* **Master Catalog and configuration** e.g. You can store all the application default or user based configuration settings in respective strings key e.g.  
    
  SET app:config:website www.test.com
* The Redis String type is the simplest type of value you can associate with a Redis key.
* String is not the best term for this type because it is used to hold numeric types as well.
* Internally it is held as a byte array.
* A string is a simple scalar that can hold a
  + single value or
  + it can hold an XML or
  + a JSON document as well
* A string value can't be bigger than 512 MB.

**SET, SETNX, MSET, MSETNX**

Set value, Set value if key does not exist, Set multiple values, Set multiple values if none of the keys exist

**GET, MGET, GETSET**

Get value, Get multiple values, Set value, return old value

**PSETEX, SETEX**

Set with expiry in seconds and milliseconds

**INCR, INCRBY, INCRBYFLOAT**

Increment integer, Add to integer, Add to float

**DECR, DECRBY**

Decrement integer, Subtract from integer

**Use Cases - Lists**

Lists contain strings that are sorted by their insertion order. With Redis Lists, you can add items to the head or tail of the lists, which is very useful for queueing jobs. Some of use cases for the list may be;

1. **Social Networking Sites:**Social platforms like Twitter use Redis Lists to populate their timelines or homepage feeds, and can customize the top of their feeds with trending tweets or stories.
2. **RSS Feeds:**We can create news feeds from custom sources where you can pull the latest updates and allow interested followers to subscribe to their favorite RSS feed.
3. **Leaderboards:** This is more of high use cases where Forums like Reddit and other voting platforms utilize Redis Lists to add articles to the leaderboard and even sort them by most viewed and voted entries.

**Quick Reference - Lists**

* A list is just a sequence of ordered elements.
* What's the downside? Accessing an element *by index* is very fast in lists implemented with an Array (constant time indexed access) and not so fast in lists implemented by linked lists (where the operation requires an amount of work proportional to the index of the accessed element).
* You can think of list as an array
* List are designed in such a way that adding new elements at the end of a list, is really fast.
* The downside is that indexing into the list can be slow.
* When this is required, Sorted Sets are a better option

**LPUSH, RPUSH**

Add Value at beginning, Add value at the end

**LPUSHX, RPUSHX**

Only push if key already exists

**LLEN, LRANGE**

Get number of values, Get values from Start to Stop

**LINDEX, LSET, LINSERT**

Get a value by index, Set a value by index, Add a value before or after another

**LREM, LTRIM**

Delete element from list, Trim list by range

**LPOP, RPOP**

Delete and Get the first element, Delete and Get the last element

**Use Cases - Hashes**

Redis Hashes are maps between string fields and string values. They are the go-to data type if you need to essentially create a container of unique fields and their values to represent objects.

1. **User Profiles:** Redis Hash data structures can be used to store application objects like Users information and more. Many web applications use Redis Hashes for their user profiles, as they can use a single hash for all the user fields, such as name, surname, email, password, etc.e.g. you can use  
     
   **HSET** user:101 name "Joe" age 30 country "USA"
2. **User Posts:** Some of the popular social platforms like instagram uses Redis Hashes for various purposes e.g.
   1. To map all the archived user photos or posts back belong to  a single user. The Redis Hashes hashing mechanism allows them to look up and return values very fast, fit the data needed in memory, and leverage data persistence in the event one of their servers dies.
3. **Storing Multi-Tenant Metrics:**Multi-tenant metrics can utilize Redis hashes to store critical informations e.g They can use hash data structures to record and store their product and sales metrics in a way that guarantees solid separation between each tenant, as hashes can be encoded efficiently in a very small memory space.

You should use Redis Hashes whenever possible, as you can use a small Redis instance to store millions of objects.

**Quick Reference - Hashes**

* hashes are useful for representing objects
* Hashes contain one or more fields.

**HSET, HSETNX, HMSET**

Set field value, Set field value if field does not exist, Set multiple field values

**HGET, HMGET**

Get field value, Get multiple field values

**HLEN, HKEYS, HVALS, HGETALL**

Get Number of fields, Get all field keys, Get all field values, Get all fields and values

**HEXISTS, HDEL**

Check field exists, delete field

**HINCRBY, HINCRBYFLOAT**

Increment field integer value, Increment field float value

Redis Sets can be used in a lot of scenarios like;

* Unique user tracking visiting a website
* Holding unique list of items like user groups, user avtar names, categories, products names, country codes etc.
* Sharded data of unique values for an application
* IP Tracking - Storing unique IP addresses to track visitors
* Implementing a product recommendation based on a user action, similar to Amazon feature where they display 'People also buy these items'
* Analyzing Ecommerce Sales - Many online e-commerce websites use Redis Sets to analyze customer behavior, such as searches or purchases for a specific product category or subcategory. For example, an online bookstore owner can find out how many customers purchased technology books in Technology section.
* Inappropriate Content Filtering - For any app that collects user input, it’s a good idea to implement some kind of content filtering for any inappropriate words, and we can do this with Redis Sets by adding words you’d like to filter to a SET key and the SADD command. e.g.  
    
  SADD bad\_words "word1" "word2"

**Quick Reference - Sets**

* Redis Sets are unordered collections of strings.
* They cannot have duplicate values
* Sets are good for expressing relations between objects

**SADD, SMOVE, SREM, SPOP**

Add one or more members, Move a member from one set to another, Remove one or more members, Remove and return one or multiple random members

**SCARD, SMEMBERS, SISMEMBER, SRANDMEMBER**

Get number of members, Get all members, Test if member exists, Get one or more random members

**SUNION, SUNIONSTORE**

Get all keys from all sets, no duplicates, same operation but store results in a new key

**SINTER, SINTERSTORE**

Get keys that exist in all sets only, same operation but store results in a new key

**SDIFF, SDIFFSTORE**

Return keys from the first set that are not in the subsequent sets, same operation but store results in a new key

**Use Cases - Sorted Sets**

You can call sorted sets a superset version of sets as it carries all the features of sets, only that sets have no order while sorted sets associate every member with a numeric score.

* Multiplayer Gaming: On an online gaming applications which required to update multiple games scores list and frequently load top score performers.
* Questions and Answers / Community forums: Redis.io website uses Sorted Sets to handle to rank the highest voted answers for each question. In addition stack overflow uses a similar setup for their Q&A setups.

**Quick Reference - Sorted Sets**

* Sorted sets are a data type which is similar to a mix between a Set and a Hash
* Sorted sets are implemented via a dual-ported data structure containing both a skip list and a hash table
* Adding elements and retrieving sorted elements is extremely fast.
* Like Hashes, sorted sets store multiple fields called members, and their numerical values called scores
* All the members are always unique
* They are ordered based on their scores

**ZADD, ZINCRBY**

Add one or more members or update score, Increment the score of a member, Remove one or more members

**ZCARD, ZCOUNT, ZSCORE, ZRANK**

Get number of members, Count members within sort key (score) range, Get the score associated with the given member, Determine the index of a member

**ZRANGE, ZRANGEBYLEX, ZRANGEBYSCORE, ZLEXCOUNT**

Get members sorted by sort key (score), Return a range of members by lexicographical range, Return a range of members by score, Count the number of members between a given lexicographical range

**ZREM, ZREMRANGEBYLEX, ZREMRANGEBYRANK, ZREMRANGEBYSCORE**

Remove one or more members, Remove all members between the given lexicographical range, Remove all members within the given indexes, Remove all members within the given scores

**ZREVRANGE, ZREVRANGEBYSCORE, ZREVRANK, ZREVRANGEBYLEX**

Return a range of members by index with scores ordered from high to low, Return a range of members by score, with scores ordered from high to low, Determine the index of a member with scores ordered from high to low, Return a range of members by lexicographical range, ordered from higher to lower strings

**ZINTERSTORE, ZUNIONSTORE**

Get keys that exist in all sets only and store the resulting sorted set in a new key, Add multiple sorted sets and store the resulting sorted set in a new key

**Use Cases - HyperLogLog**

To get unique data usually you need more memory, since bigger the input data, more are the chances of getting a stable output. HyperLogLog is good at this since you get great performance at low computation cost, and a small amount of memory. HyperLogLog can be used in various situations like;

* Counting unique visitors
* Creating unique in a book or books
* Keeping best stocks of all times
* Create unique names of a products, services, category
* A situation where you are not worries about the elements of the datasets, but its counts
* Create unique names of students in a class and many more.

**Quick Reference - HyperLogLog**

* A HyperLogLog is a probabilistic data structure
* Used in order to count unique things (technically this is referred to estimating the cardinality of a set).
* Encoded as a Redis string
* Conceptually the HLL (HyperLogLog) is like using Sets to do the same task.
* They are not one hundred percent accurate
* By using HyperLogLogs we can save a tremendous amount of memory.

**PFADD, PFCOUNT**

Append one or more elements, count number of elements

**PFMERGE**

Merge elements from multiple keys

**Unsubscribe from a channel or channels**

**UNSUBSCRIBE** Channel Name

Unsubscribe channel name command will allow the client to unsubscribe to a particular channel. e.g. if you have a channel called news1 and you want to unsubscribe then you will run the following command;

**> UNSUBSCRIBE** news1  
  
Please note that as you had seen from the earlier videos that, when you are running the SUBSCRIBE command from the redis-cli, the only option for you to unsubscribe from the client is to run CRTL -C to quit. However lets say if you are interacting with Redis with a programming language like Python etc., then you can unsubscribe from the client with the above command.

**What about unsubscribe from all channels?**You can use just simply run UNSUBSCRIBE without any channel name and client will be unsubscribed from all channels.

**What if a client wants to unsubscribe channels based on a patterns e.g. unsubscribed from all news channels?**

You can use the following command to do that;

**PUNSUBSCRIBE** <pattern>  
i.e.  
  
>**PUNSUBSCRIBE** news\*

Please note, when no patterns are specified, the client is unsubscribed from all the previously subscribed patterns. In this case, a message for every unsubscribed pattern will be sent to the client.

Happy Pub/Subbing!!!